2018年10月21日 星期日

Compte Rendu IA01_TD05

Objectif: programmer un moteur d'ordre 0+ en utilisant une stratégie de type chaînage arrière.

Ordres

Moteur d'ordre 0 : on travaille avec des propositions booléenne

Moteur d'ordre 0+ : les faits peuvent avoir des valeurs

Moteur d'ordre 1 : introduction de variables locales

Chaînage-avant et chaînage-arrière

Faire une simulation papier d'une recherche en chaînage-avant, puis e chaînage-arrière, avec les données suivantes:

Base de règles:

R1	B et D et E ->F
R2	D et G ->A
R3	C et F ->A
R4	C ->D
R5	D ->E
R6	A ->H
R7	B ->X
R8	X et C ->A

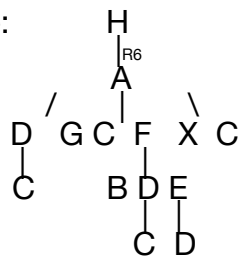
Base de faits : B, C
But: H

En chaînage-avant:

Mon résultat(faux): B,C,D,E,F,A,H

Résultat vrai: B,C,D,X,A,H

En chaînage-arrière:



Représentation

Proposer une représentation de la base de règles et de la base de faits.

Pour Base de règles: on implémente la base de règles avec *BR* qui est une liste des règles où les règles sont représentées sous la forme :

(numDeLaRegle (premisse1 … premisseN) resultat)

```
(setq *BR* '(
            (R1 (B D E) F)
            (R2 (D G) A)
            (R3 (C F) A)
            (R4 (C) D)
            (R5 (D) E)
            (R6 (A) H)
            (R7 (B) X)
            (R8 (X C) A)
          )
)
```

Pour Base de faits : on liste simplement dans *BF* les faits.

```
(setq *BF* '(B C))
```

Algorithme

Écrire un algorithme réalisant un chaînage en arrière en profondeur d'abord.

Fonctions des services:

```
(defun CCLRegle (regle)
    (caddr regle)
)
(defun premisseRegle (regle)
    (cadr regle)
)
(defun nunmRegle(regle)
    (car regle)
```

```
)

(defun ReglesCandidates(But bdR)
    (let ( ( candidates '() ) )
        (loop for x in bdR
                do (if (equal But (CCLRegle x) )
                        (push  x candidates)
                       nil
                    )
            )
        (print candidates)
     )
    ;;     (dolist (x bdR candidates)
    ;;         (if (equal But (CCLRegle x))
    ;;         (push x candidates))
    ;;     )
    ;;     (print candidates)
    ;; )
)
(defun ReglesCandidatesR (But bdR) ;;fonction en récursif
    (if bdR
        (if (equal But (CCLRegle (car bdR)))
            (cons (car bdR) (ReglesCandidatesR But (cdr bdR)))
            (ReglesCandidatesR But (cdr bdR))

        )
    )
)

; Fonctions principales

(defun verifier (but) ; c'est elle que l'on lance en premier
  (or
    (member but *BF*)
    (let (ensConflits (reglesCandidates but *BR*) OK)
      (loop
        (if (or OK (null ensConflits))
          (return nil)
```

```
        )
        (setq OK (verifierET (pop ensConflits)))
      )
      OK
    )
  )
)
```

```
(defun verifierET (regle)
  (let ((OK t) (premisses (termesConditions regle)))
    (loop
      (if (or (not OK)(null premisses))
        (return nil)
      )
      (setq OK (verifier (pop premisses)))
    )
    OK
  )
)
```

Pop: elever le premier élément de la liste.